EECE5639
COMPUTER VISION

PROJECT 2

IMAGE MOSAICING

SHIVA KUMAR DANDE
SAI PRASASTH KOUNDINYA GANDRAKOTA

ABSTRACT

The objective is to detect corners in two images by applying a Harris corner detector, automatically match corresponding features, estimate a homography that relates the two images, and then transform one image to fit the coordinate system of the other. The final output will be a mosaic that includes all the pixels present in both images.

DESCRIPTION OF ALGORITHMS

HARRIS CORNER DETECTOR

Firstly, Harris Corner Detector algorithm was applied. The input images were converted to greyscale and then both vertical and horizontal Sobel masks were applied to each image. Then the M matrix was calculated from the resultant I_x^2, I_y^2 and I_x*I_y values. The Harris R function value was then determined from the M matrix using the formula $R = \det(M) - k*[\text{trace}(M)]^2$ where $\det(M) = \lambda_1 * \lambda_2$, $\text{trace}(M) = \lambda_1 + \lambda_2$ and k value is 0.04. A high threshold of 1000000 was set to the R matrix of all images and non-max suppression was applied to the R function to obtain a sparse set of corner features.

NORMALIZED CROSS CORRELATION

In order to detect the corner matches between the input images the normalized cross correlation matrix was computed between each image pair, using patches of the image with each corner at the center. Pairs of corners with high NCC values were isolated and then applied against a threshold of 0.8 in order to determine the matched corner features between the image pairs.

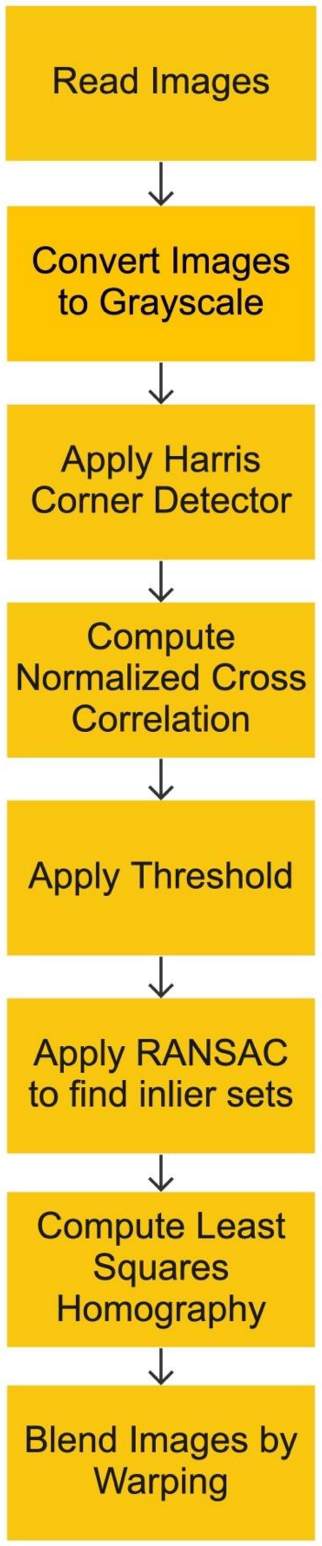
HOMOGRAPHY MATRIX (RANSAC)

A homography matrix was then computed using a 4-point RANSAC method over one thousand iterations. The distance between the inliers and the sample points was calculated and the number of inliers was derived as well. Subsequently the least-squares homography matrix between each image pair was then obtained.

IMAGE WARPING (MOSAICING)

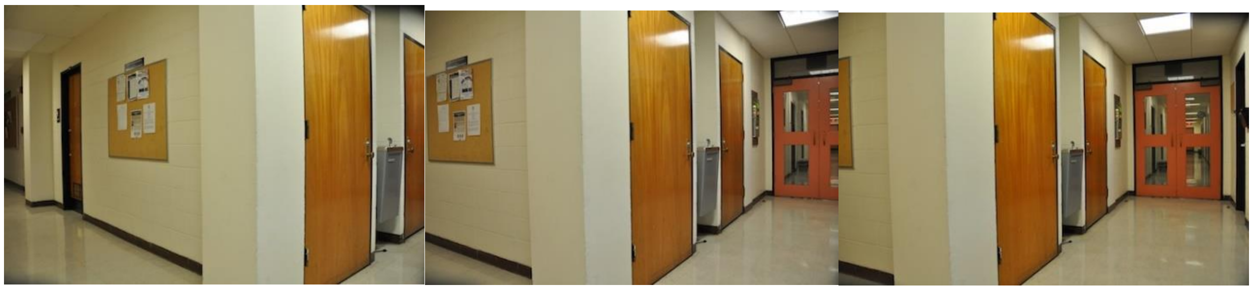
The first step in Image Warping involved determining the size of the final output image, which needed to be large enough to contain all pixels from the input images. Then the frame of reference image is copied into the output image at the appropriate location using a translation matrix. Then the next images is warped onto the output image using the calculated homography matrices and the imwarp() function. Finally, the overlapping areas of the images were blended together by taking the maximum intensity value between the two pixels. The result is a single, merged image that shows the union of all pixels from the input images. The result was then again applied through the above process with the next image and so forth.

FLOWCHART


Read Images
Convert Images to Grayscale
Apply Harris Corner Detector
Compute Normalized Cross Correlation
Apply Threshold
Apply RANSAC to find inlier sets
Compute Least Squares Homography
Blend Images by Warping

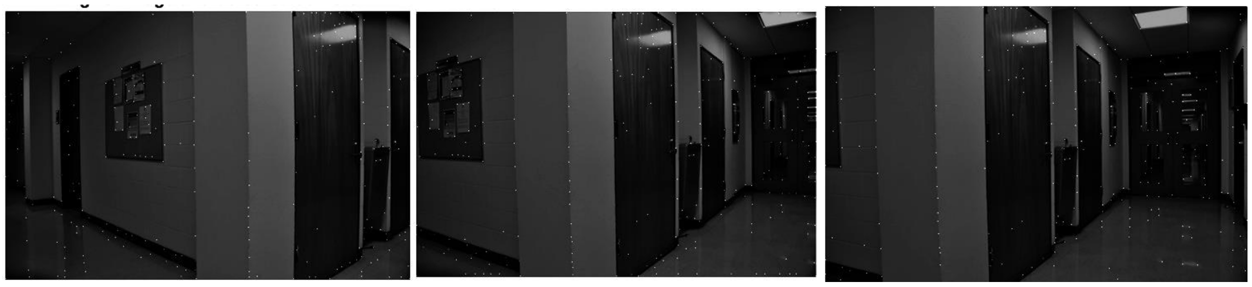
EXPERIMENTS

INPUT IMAGES

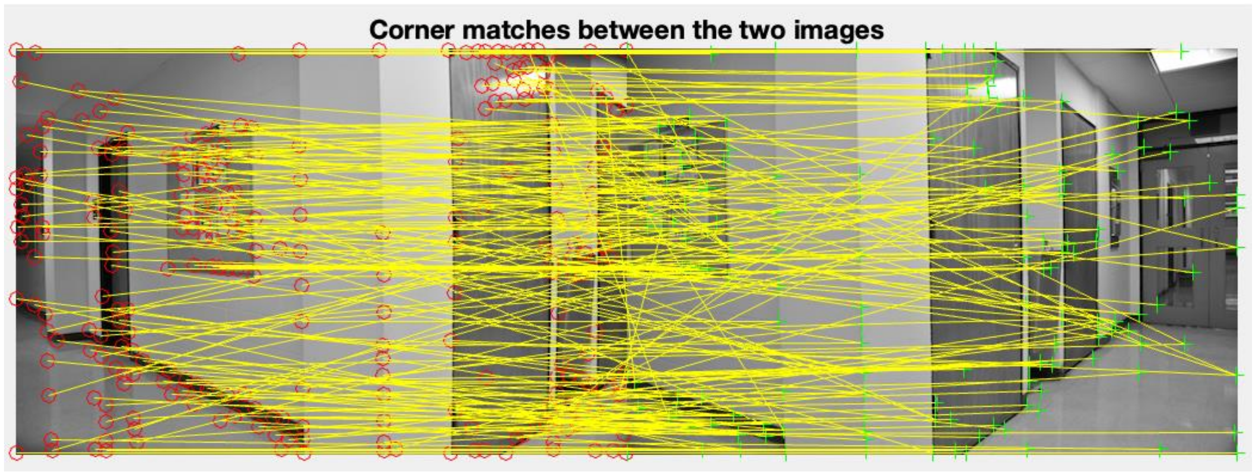


GRAYSCALE IMAGES

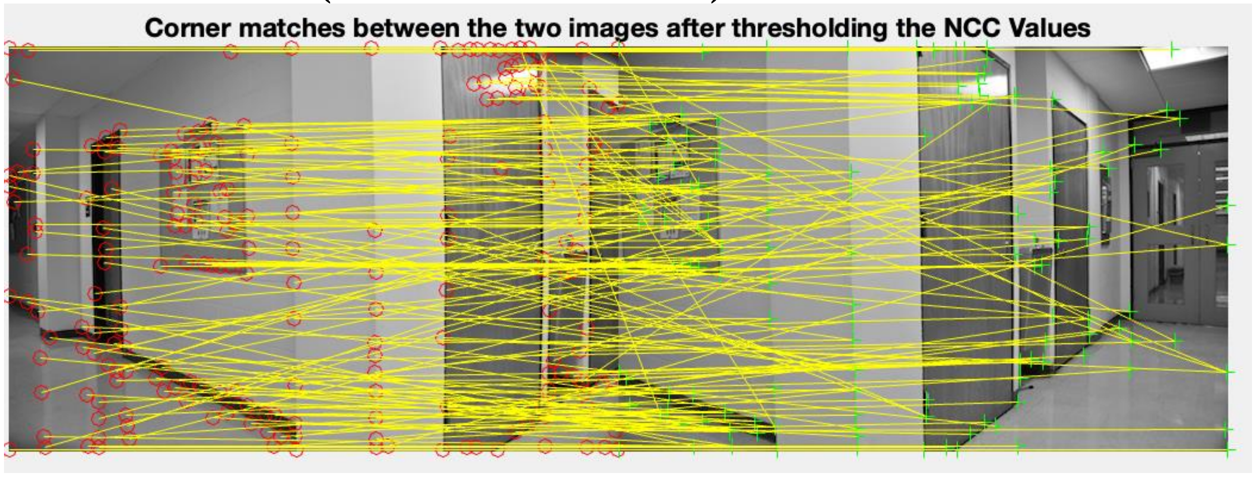
DETECTED CORNERS



CORNER MATCHES



CORNER MATCHES (AFTER THRESHOLDING)


Corner matches between the two images after thresholding the NCC Values

CORNER MATCHES (AFTER RANSAC)

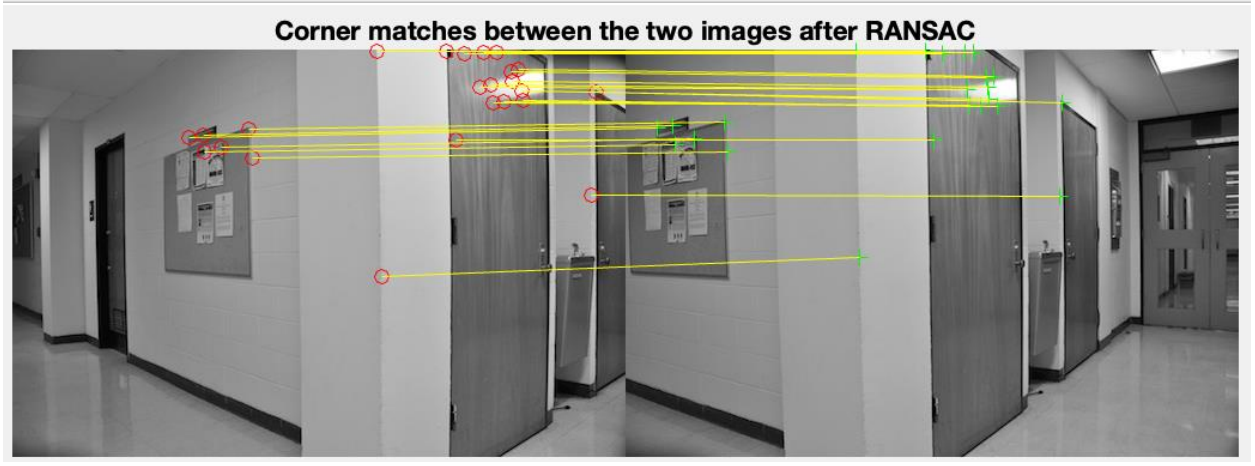
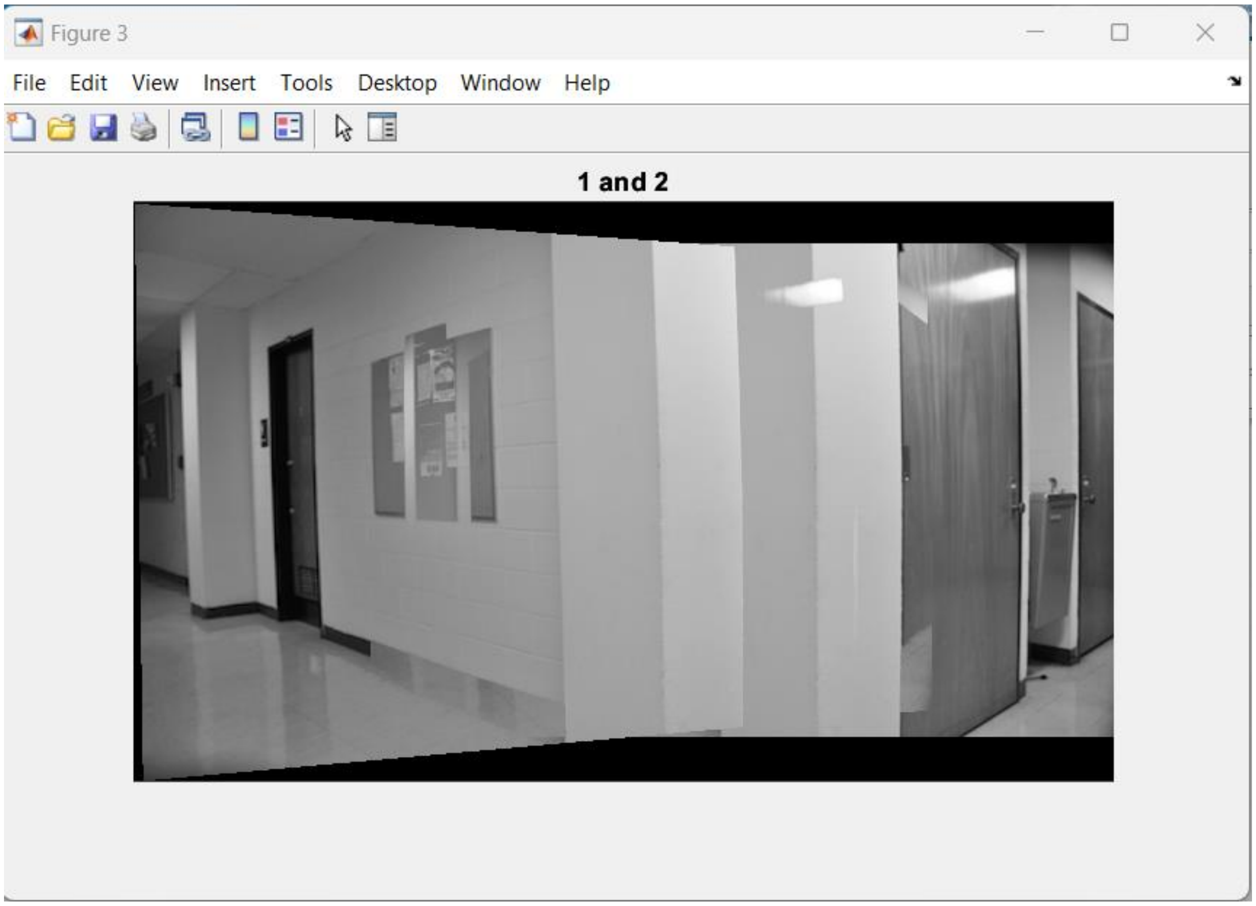
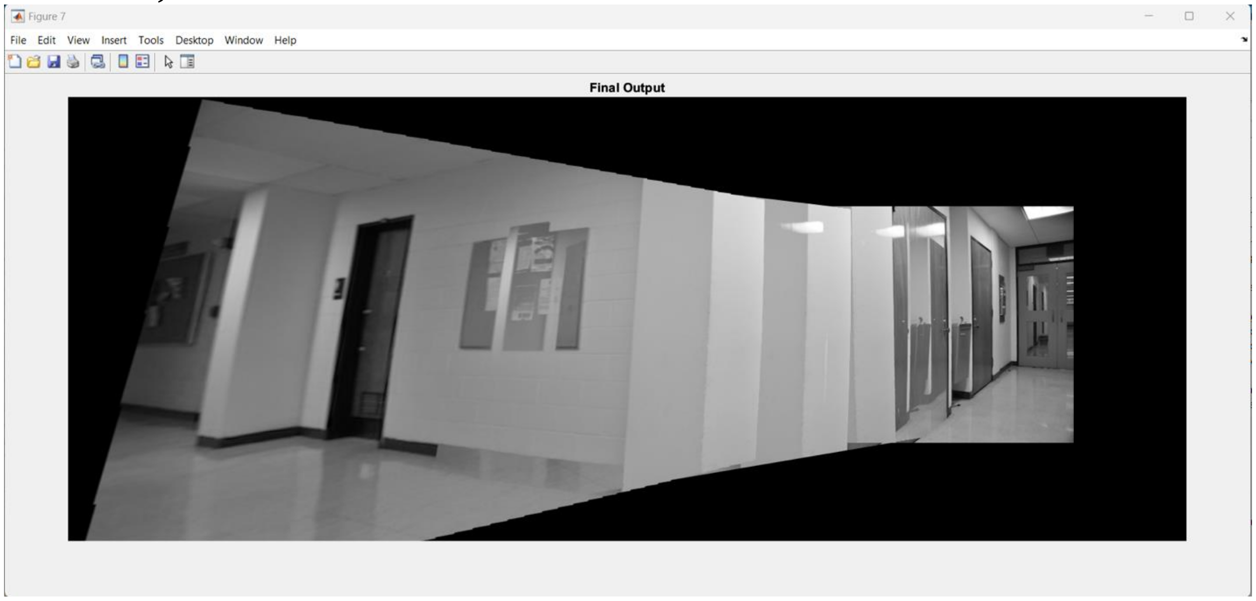


IMAGE WARPING (BLENDING)

IMAGE 1 AND 2 WARPED



IMAGES 1, 2 AND 3 WARPED



PARAMETERS USED

- The images read in were both of sizes 340x512 pixels.
- In the Harris Corner Detection algorithm, we filter the images with a Prewitt Filter to obtain the C matrix and then a Gaussian filter of size 3x3 is applied to the C matrix.
- The threshold for non-maximum suppression was 1000000 and the window size was 11x11.
- After finding the corner sets in both images using the computed normalized cross correlation using window patches of size 9x9.
- The number of corner pairs was reduced after the threshold of 0.75 was applied to the NCC values.
- For RANSAC, the maximum number of iterations set was 72, and after trial and error a max distance of 20 was chosen to classify inliers and outliers.

OBSERVATIONS AND CONCLUSIONS

- In the Harris Corner Detection algorithm, the Gaussian filter applied to the C matrices helps in reducing the noise and improving the robustness of the algorithm.
- The size of the patches being created determines the efficiency and precision of calculating the normalized cross correlation which in turn could lead to unwanted feature matches or instead missed feature matches.
- The threshold for NCC affects the number of potential corner matches in the images. Setting a high value would result in very few matches being detected and risks missing out on features, whereas low threshold values would result in non-features being detected as potential matches as well.
- For RANSAC, as we increase the maximum distance the number of inliers detected increases and ends up giving an unfavorable outcome. However, for low values of the maximum distance we might not be able to determine more than a couple corner points which would result in the process stopping as we need at least 4 point pairs to calculate the homography matrices.
- It is possible to blend two images into a single image using warping provided they have enough features that can be matched.

APPENDIX

CODE (MATLAB)

```
clear all;
close all;
clc;
%path of the image directory
path = "/Users/prasasth/Downloads/DanaHallWay1";
%reading and displaying images
input_images = read_images_from_path(path);
% display_images(input_images, 'Colour  Images');
%converting images to gray scale
gray_images = convert_grayScale(input_images);
TFORM = [];
numImages = size(input_images, 4);
imageSize = []; %zeros(numImages, 2);
warped_image = gray_images(:, :, :, 1);

for i = 1:size(gray_images, 4) - 1
    % display_images(gray_images, 'Gray Scaled Images');
    [r1, c1] = harris_corner_alg(warped_image);
    [r2, c2] = harris_corner_alg(gray_images(:, :, :, i+1));
    row_col1 = [r1, c1];
    row_col2 = [r2, c2];
    [max_ncc_values, matched_corner_sets] = norm_cross_corr(row_col1, warped_image, row_col2,
gray_images(:, :, :, i+1));
    ncc_threshold = 0.75;
    [ncc_th, matched_corner_sets_th] = threshold_ncc(max_ncc_values, matched_corner_sets,
ncc_threshold);
    corners_image1 = flip(matched_corner_sets(:, 1:2), 2);
    corners_image2 = flip(matched_corner_sets(:, 3:4), 2);
    figure;
    showMatchedFeatures(warped_image, gray_images(:, :, :, i+1), corners_image1, corners_image2,
'montage');
    title('Corner matches between the two images');
    waitforbuttonpress;
    close;
    corners_image1_th = flip(matched_corner_sets_th(:, 1:2), 2);
    corners_image2_th = flip(matched_corner_sets_th(:, 3:4), 2);
    figure;
    showMatchedFeatures(warped_image, gray_images(:, :, :, i+1), corners_image1_th,
corners_image2_th, 'montage');
    title('Corner matches between the two images after thresholding the NCC Values');
    waitforbuttonpress;
    close;
    [inlier_sets, tform] = RANSAC(matched_corner_sets_th);
%     tform.T
%     TFORM(:, :, i) = tform.T';
    figure;
    showMatchedFeatures(warped_image, gray_images(:, :, :, i+1), flip(inlier_sets(:, 1:2), 2),
flip(inlier_sets(:, 3:4), 2), 'montage');
    title('Corner matches between the two images after RANSAC');
    waitforbuttonpress;
    close;
    warped_image = warp_images(warped_image, gray_images(:, :, :, i), tform);
end

figure;
imshow(warped_image);
title("Final Output");

function img_combined = warp_images(img1, img2, H)
        homography = H.T';
        homography([1 2], :) = homography([2 1], :);
        homography(:, [1 2]) = homography(:, [2 1]);
        % Compute dimensions of the resulting image
        [w1, h1, ~] = size(img1);
        [w2, h2, ~] = size(img2);
        corners = single([0, 0; 0, h2; w2, h2; w2, 0]);
        corners_transformed = transformPointsForward(H, corners);
        x_min = min(min(corners_transformed(:, 1)), 0);
        x_max = max(max(corners_transformed(:, 1)), w1);
        y_min = min(min(corners_transformed(:, 2)), 0);
        y_max = max(max(corners_transformed(:, 2)), h1);
        output_shape = [round(x_max - x_min), round(y_max - y_min)];
        % Compute translation matrix to shift the transformed image into place
        translation_matrix = [1, 0, -y_min; 0, 1, -x_min; 0, 0, 1];

        % Warp images using the homography matrix and translation matrix
        img1_warped = imwarp(img1, projective2d((translation_matrix * homography)'), 'OutputView',
imref2d(output_shape));
        img2_warped = imwarp(img2, projective2d(translation_matrix'), 'OutputView',
imref2d(output_shape));
        img_combined = max(img1_warped, img2_warped);
        figure;
        imshow(img_combined);
        title('Combined Image (2 Images)');
        waitforbuttonpress;
        close;
end

function homography_matrix = compute_homography(src, dst, i)

    x1 = src(1,1); y1 = src(1,2);
    x2 = dst(1,1); y2 = dst(1,2);
    x3 = src(2,1); y3 = src(2,2);
    x4 = dst(2,1); y4 = dst(2,2);
    x5 = src(3,1); y5 = src(3,2);
    x6 = dst(3,1); y6 = dst(3,2);
    x7 = src(4,1); y7 = src(4,2);
    x8 = dst(4,1); y8 = dst(4,2);

    % Construct the matrix A
    A = [-x1, -y1, -1, 0, 0, 0, x2*x1, y2*x1, x2;
     0, 0, 0, -x1, -y1, -1, x2*y1, y2*y1, y2;
     -x3, -y3, -1, 0, 0, 0, x4*x3, y4*x3, x4;
     0, 0, 0, -x3, -y3, -1, x4*y3, y4*y3, y4;
     -x5, -y5, -1, 0, 0, 0, x6*x5, y6*x5, x6;
     0, 0, 0, -x5, -y5, -1, x6*y5, y6*y5, y6;
     -x7, -y7, -1, 0, 0, 0, x8*x7, y8*x7, x8;
     0, 0, 0, -x7, -y7, -1, x8*y7, y8*y7, y8];
    % Compute the null space of A using singular value decomposition
    [U, S, V] = svd(A);
    h = U(:,end);
    h(end+1) = 1;
    homography_matrix = reshape(h,3,3)';
    if (i == 1)
        h = V(:, end);
        homography_matrix = reshape(h,3,3)';
    end
end
function predicted_dst_points = apply_homography(homography_matrix, src_points)
    H = homography_matrix;
    predicted_dst_points = transformPointsForward(projective2d(H'), src_points);
end

function num_inliers = compute_inliers(dst_points, predicted_dst, maxDistance)
    distances = sqrt(sum((predicted_dst - dst_points).^2, 2));
    % find inliers based on distance threshold
    num_inliers = sum((distances < maxDistance));
end

function tform = compute_least_sq_homography(src_pts,dst_pts)
    n = size(src_pts,1);
    A = zeros(2*n,9);
    tform = zeros(3);
    for i = 1:n
        xsi = src_pts(i,1);
        ysi = src_pts(i,2);
        xdi = dst_pts(i,1);
        ydi = dst_pts(i,2);
        A((2*i)-1,:) = [xsi,ysi,1,0,0,0,-(xdi*xsi),-(xdi*ysi),-(xdi)];
        A(2*i,:) = [0,0,0,xsi,ysi,1,-(ydi*xsi),-(ydi*ysi),-(ydi)];
    end
    [~, ~, V] = svd(A);
    h = V(:, end);
    tform = reshape(h,3,3)';
    tform = tform/ tform(3,3);
end
function [inlier_sets, tform] = RANSAC(matched_corner_sets_th)
    iterations = 72;
    maxDistance = 20;
    max_inliers = 0;
    src_points = [matched_corner_sets_th(:, 1:2)];
    dst_points = [matched_corner_sets_th(:, 3:4)];
%     [homography_matrix, inliers] = estimateGeometricTransform(points1, points2, 'projective',
'MaxNumTrials', 2000, 'Confidence', 99);
    for i = 1 : iterations
        indices = randperm(size(matched_corner_sets_th, 1), 4);
        points = matched_corner_sets_th(indices, :);
        src = [points(:, 1:2)];
        dst = [points(:, 3:4)];
        homography_matrix = compute_homography(src, dst, 1);
        predicted_dst = apply_homography(homography_matrix, src_points);
        num_inliers = compute_inliers(dst_points, predicted_dst, maxDistance);
        if (num_inliers > max_inliers)
            max_inliers = num_inliers;
            best_Homography = homography_matrix;
        end
    end
    disp(best_Homography);
    %using the best homography finding the inliers
    predicted_dst = apply_homography(best_Homography, src_points);
    distances = sqrt(sum((predicted_dst - dst_points).^2, 2));
    % find inliers based on distance threshold
    inliers_indices = distances < maxDistance;
    inlier_sets = matched_corner_sets_th(inliers_indices, :);
    src_points = [ inlier_sets(:, 1:2) ];
    dst_points = [ inlier_sets(:, 3:4) ];
    tform1 = compute_least_sq_homography(src_points, dst_points);
    tform = estgeotform2d(dst_points, src_points,'projective');
end
function [ncc_th, matched_corner_sets_th] = threshold_ncc(max_ncc_values, matched_corner_sets,
threshold)
    ncc_th = [];
    matched_corner_sets_th = [];
    for i = 1 : length(max_ncc_values)
        if(max_ncc_values(i) >= threshold)
            %Appending if max_ncc_value is greater than threshold
            ncc_th(end+1) = max_ncc_values(i);
            matched_corner_sets_th(end+1, :) = matched_corner_sets(i, :);
        end
    end
end
function image_patch = create_image_patch(row_center, col_center, image, w_size)
    img_size = size(image);

    if(row_center - (w_size-1)/2 > 1)
        start_i = row_center - (w_size-1)/2;
    else
        start_i = 1;
    end

    if(col_center - (w_size-1)/2 > 1)
        start_j = col_center - (w_size-1)/2;
    else
        start_j= 1;
    end
    if(row_center + (w_size-1)/2 < img_size(1))
        end_i = row_center + (w_size-1)/2;
    else
        end_i = img_size(1);
    end

    if(col_center + (w_size-1)/2 < img_size(2))
        end_j = col_center + (w_size-1)/2;
    else
        end_j = img_size(2);
    end
    % Creating patches
    if (size(image, 3) == 3)
        gray_image = rgb2gray(image);
    else
        gray_image = image;
    end
%     image_patch = gray_image(start_i : end_i, start_j : end_j);
    image_patch = gray_image(start_i : end_i, start_j : end_j);
    required_patch_size = [w_size w_size];
    patch_size = size(image_patch);
    if (patch_size(1) ~= required_patch_size(1))
        r_size = w_size - patch_size(1);
        padSize = [r_size, 0];
        if (row_center - (w_size-1)/2 < 1)
            image_patch = padarray(image_patch, padSize, 0, 'pre');
        elseif (row_center + (w_size-1)/2 > img_size(1))
            image_patch = padarray(image_patch, padSize, 0, 'post');
        end
    end
    if (patch_size(2) ~= required_patch_size(2))
```

```
        c_size = w_size - patch_size(2);
        padSize = [0, c_size];
        if (col_center - (w_size-1)/2 < 1)
            image_patch = padarray(image_patch, padSize, 0, 'pre');
        elseif (col_center + (w_size-1)/2 > img_size(2))
            image_patch = padarray(image_patch, padSize, 0, 'post');
        end
    end
    image_patch = im2double(image_patch);
end
function [max_ncc_values, matched_corner_sets] = norm_cross_corr(row_col1, image1, row_col2,
image2)
    w_size = 9;
    max_ncc_values = [];
    matched_corner_sets = [];
    img1_corner = zeros(1,2);
    img2_corner = zeros(1,2);
    for i = 1 : length(row_col1)
        row_center_img1 = row_col1(i, 1);
        col_center_img1 = row_col1(i, 2);
        image1_patch = create_image_patch(row_center_img1, col_center_img1, image1, w_size);
        mean_img1_patch = mean(image1_patch( : ));
        max_ncc = 0;
        for j = 1:length(row_col2)
            row_center_img2 = row_col2(j, 1);
            col_center_img2 = row_col2(j, 2);
            image2_patch = create_image_patch(row_center_img2, col_center_img2, image2, w_size);
            mean_img2_patch = mean(image2_patch( : ));
            product = (image1_patch - mean_img1_patch) .* (image2_patch - mean_img2_patch);
            numerator_term = sum( product( : ) );

            s1 = (image1_patch - mean_img1_patch) .^ 2;
            s1 = sum( s1( : ) );
            s2 = (image2_patch - mean_img2_patch) .^ 2;
            s2 = sum( s2( : ) );
            denominator_term = sqrt(s1 * s2);
            ncc = numerator_term / denominator_term;
            if (max_ncc < ncc)
                img1_corner = [row_center_img1, col_center_img1];
                img2_corner = [row_center_img2, col_center_img2];
                max_ncc = ncc;
            end
        end
        max_ncc_values(i) = max_ncc;
        matched_corner_sets(i, :) = [img1_corner, img2_corner];
    end
end
function [r, c] = harris_corner_alg(image)
```

```
    % checks if the image is coloured and converts to gray
    if (size(image, 3) == 3)
        gray_image = rgb2gray(image);
    else
        gray_image = image;
    end

    %Sobel masks
    sobel_X = [-1, -2, -1; 0, 0, 0; 1, 2, 1];
    sobel_Y = [-1, 0, 1; -2, 0, 2; -1, 0, 1];

    % Calculating Ix, Iy, Ix2, Iy2, Ix.*Iy
    Ix = conv2(gray_image, sobel_X, 'same');
    Iy = conv2(gray_image, sobel_Y, 'same');

    %The use of the Gaussian filter in Harris corner detector helps to improve
    % the accuracy and robustness by reducing the noise of the corner detection
    % algorithm.
    size_window = 3;
    gaussian_filter = gausswin(size_window.^2);
    Ix2 = conv2(Ix .^ 2, gaussian_filter, 'same');
    Iy2 = conv2(Iy .^ 2, gaussian_filter, 'same');
    Ixy = conv2(Ix .* Iy, gaussian_filter, 'same');

    %M = [ Ix2  Ixy ]
    %    [ Ixy  Iy2 ]
    %Calculating R value (det(M) - k * sq(trace(M)))
    det = (Ix2 .* Iy2) - (Ixy .* Ixy);
    trace = Ix2 + Iy2;
    k = 0.04;
    R = det - k*(trace .^ 2);

    % non-max suppression

ow = 11;
    threshold = 1000000;
    median_filtered = ordfilt2(R, size_window^2, ones(size_window));

    % threshold
    harris = (R == median_filtered) & (R> threshold);

    % Detect corners using the Harris corner detector
    % Uncomment the below line to use inbuild Harris Corner Detector Function
    %corners = detectHarrisFeatures(gray_image);

    % Finding the row and column number of the image where the harris value is
    % greater than or equal to one
    [r, c] = ind2sub(size(harris), find(harris >= 1));
    % Display the image with the detected corners
    imshowpair(image, harris, 'blend');
    title("original image and detected corners");
    waitforbuttonpress;
    close;
end
function gray_images = convert_grayScale(images)
    gray_images = [];
    for i = 1:size(images, 4)
        image = im2gray(images(:, :, :, i));
        gray_images = cat(4, gray_images, image);
    end
end

function display_images(images, image_Title)
    %This function displays the images
    for i = 1:size(images, 4)
        imshow(images(:, :, :, i));
        title(image_Title);
        pause(0.01);
    end
    waitforbuttonpress;
    close;
end
function images = read_images_from_path(path)
    % This function reads in images from a specified path and returns them as
    % an array of images.

    % List all files in the specified path
    file_list = dir(path);
    % Initialize an empty array to store the images
    images = [];

    % Loop through each file in the path
    for i = 1:length(file_list)
        % Get the current file name
        file_name = file_list(i).name;

        % Check if the current file is an image file
        if endsWith(file_name, {'.JPG', '.jpeg', '.png', '.bmp', '.gif'})
            % Read in the image and add it to the array of images
            image = imread(fullfile(path, file_name));
            images = cat(4, images, image);
        end
    end
    % Display a message if no images were found
    if isempty(images)
```

```
        disp('No images found in specified path.')
    end
end
```

Code for Extra Credit:

Extra Credit Warp one image into a “frame” region in the second image. To do this, let the points from the one view be the corners of the image you want to insert in the frame and let the corresponding points in the second view be the clicked points of the frame (rectangle) into which the first image should be warped (Look at Matlab’s ginput function for an easy way to collect mouse clicks). Use this idea to replace one surface in an image with a picture of your pet or project a drawing from one image onto the street in another image, or paste a powerpoint slide on a movie screen, etc ... Be creative! Send me your result by email so I can share it with the rest of the class.

```
% Load the two images that you want to use
path = "C:\Users\Shiva Kumar Dande\Desktop\Semester 2\CV_Project\Project
2_Image_mosaicing\DanaOffice\DanaOffice";
input_images = read_images_from_path(path);
img1 = input_images(:, :, :, 1);
img2 = input_images(:, :, :, 2);

% Display the images
figure(1);
imshow(img1);
figure(2);
imshow(img2);

% Use Matlab's ginput function to collect mouse clicks for the corners
% of the image you want to insert into the frame and the corresponding
% points in the second view for the corners of the rectangle into which
% the first image should be warped.

% Collect points from image1
figure(1);
disp('Select four points in image1 (in order: top-left, top-right, bottom-right, bottom-left)');
[x1, y1] = ginput(4);

% Collect points from image2
figure(2);
disp('Select four points in image2 (in order: top-left, top-right, bottom-right, bottom-left)');
[x2, y2] = ginput(4);

% Calculate the homography matrix that maps the corners of the first
% image to the corresponding points in the second image using the
% clicked points.
H = homography(x1, y1, x2, y2);
```

```
% Use the homography matrix to warp the first image into the rectangle in
% the second image.
[img1_warped, x_min, y_min, x_max, y_max] = warp_image(img1, H);

% Display the resulting image
figure(3);
imshow(img1_warped);

% Save the resulting image
imwrite(img1_warped, 'result.jpg');

% % Define the homography function
% function H = homography(x1, y1, x2, y2)
% A = zeros(8, 9);
% for i = 1:4
% A(2*i-1,:) = [-x1(i) -y1(i) -1 0 0 0 x1(i)*x2(i) y1(i) x2(i) x2(i)];
% A(2i,:) = [0 0 0 -x1(i) -y1(i) -1 x1(i)*y2(i) y1(i)*y2(i) y2(i)];
% end
% [~, ~, V] = svd(A);
% H = reshape(V(:,end), [3,3])';
% end

function H = homography(x1, y1, x2, y2)
  % construct matrix A
  A = zeros(2*size(x1,1),9);
  for i = 1:size(x1,1)
    A(2*i-1,:) = [-x1(i) -y1(i) -1 0 0 0 x1(i)*x2(i) y1(i)*x2(i) x2(i)];
    A(2*i,:) = [0 0 0 -x1(i) -y1(i) -1 x1(i)*y2(i) y1(i)*y2(i) y2(i)];
  end

  % solve for H using SVD
  [~,~,V] = svd(A);
  H = reshape(V(:,end),3,3)';
end


% Define the warp_image function
function [img_warped, x_min, y_min, x_max, y_max] = warp_image(img, H)
% Find the corners of the image after warping
[rows, cols, ~] = size(img);
corners = [1, 1, cols, cols; 1, rows, rows, 1; 1, 1, 1, 1];
warped_corners = H * corners;
warped_corners = warped_corners ./ warped_corners(3,:);
% Find the minimum and maximum x and y coordinates after warping
x_min = floor(min(warped_corners(1,:)));
y_min = floor(min(warped_corners(2,:)));
x_max = ceil(max(warped_corners(1,:)));
y_max = ceil(max(warped_corners(2,:)));
```

```
% Create a meshgrid of the warped image coordinates
[X, Y] = meshgrid(x_min:x_max, y_min:y_max);
coords = [X(:), Y(:), ones(length(X(:)),1)]';

% Invert the homography matrix
H_inv = inv(H);

% Map the warped coordinates back to the original image
warped_coords = H_inv * coords;
warped_coords = warped_coords ./ warped_coords(3,:);
warped_x = reshape(warped_coords(1,:), size(X));
warped_y = reshape(warped_coords(2,:), size(Y));

% Interpolate the values of the original image at the warped coordinates
img_warped = zeros(size(X,1), size(X,2), size(img,3));
for i = 1:size(img,3)
    img_warped(:,:,i) = interp2(double(img(:,:,i)), warped_x, warped_y, 'linear', 0);
end
end
function images = read_images_from_path(path)
    % This function reads in images from a specified path and returns them as
    % an array of images.

    % List all files in the specified path
    file_list = dir(path);
    % Initialize an empty array to store the images
    images = [];

    % Loop through each file in the path
    for i = 1:length(file_list)
        % Get the current file name
        file_name = file_list(i).name;

        % Check if the current file is an image file
        if endsWith(file_name, {'.JPG', '.jpeg', '.png', '.bmp', '.gif'})
            % Read in the image and add it to the array of images
            image = imread(fullfile(path, file_name));
            images = cat(4, images, image);
        end
    end
    % Display a message if no images were found
    if isempty(images)
        disp('No images found in specified path.')
    end
end
```